DOCUMENT RESUME

ED 428 674 IR 019 335

| | |
|---|---|
| AUTHOR | Ip, Albert; Fritze, Paul |
| TITLE | Supporting Component-Based Courseware Development Using Virtual Apparatus Framework Script. |
| PUB DATE | 1998-06-00 |
| NOTE | 8p.; In: ED-MEDIA/ED-TELECOM 98 World Conference on Educational Multimedia and Hypermedia & World Conference on Educational Telecommunications. Proceedings (10th, Freiburg, Germany, June 20-25, 1998); see IR 019 307. Figures may not reproduce clearly. |
| PUB TYPE | Reports - Descriptive (141) -- Speeches/Meeting Papers (150) |
| EDRS PRICE | MF01/PC01 Plus Postage. |
| DESCRIPTORS | *Authoring Aids (Programming); *Computer Software Development; *Courseware; Educational Technology; Foreign Countries; Higher Education; *Hypermedia; Models |
| IDENTIFIERS | University of Melbourne (Australia); Web Pages; XML |


ABSTRACT

This paper reports on the latest development of the Virtual Apparatus (VA) framework, a contribution to efforts at the University of Melbourne (Australia) to mainstream content and pedagogical functions of curricula. The integration of the educational content and pedagogical functions of learning components using an XML compatible script, VAScript, is discussed. This approach facilitates component re-usability and administration and is based on a database model matched with an authoring system that hides the technical details while preserving the contribution of content experts to the courseware development. Topics discussed include: the VA concept; typical use scenarios, including the Learning Engine object model, National Asian Languages and Studies in Australian Schools Taskforce model, flexibility in adding data-logging functions as needed, and adaptive content delivery using a back-end database server; and technical specification of VAScript, including syntax, compulsory parameters supported by version 1.1 VA, compulsory behaviors supported by version 1.1 VA, and browser script for version 1.1. (Author/DLS)

# Supporting Component-Based Courseware Development using Virtual Apparatus Framework Script

Albert Ip
Multimedia Education Unit,
The University of Melbourne.
Email a.ip@meu.unimelb.edu.au

Paul Fritze
Multimedia Education Unit,
The University of Melbourne.
Email: p.fritze@meu.unimelb.edu.au



**Abstract**

This paper reports on the latest development of the "Virtual Apparatus Framework", a contribution to efforts at the University of Melbourne to mainstream the digital transformation of curricula. We discuss the integration of the educational content and pedagogical functions of learning components using an XML compatible script. Our approach facilitates component re-usability and administration and is based on a database model matched with an authoring system that hides the technical details while preserving the contribution of content experts to the courseware development. In particular, this paper is to be the reference point of version 1.1 of the VA framework.

## Introduction

The concept of "Virtual Apparatus" (VA), software components written to a specification and providing interactivity on a web page, was discussed and demonstrated at ASCILITE 1996 [Ip and Canale,1996]. We have since brought the implementation forward so that virtual apparatus built using different technologies (such as Java Applet, Active X controls & Macromedia Shockwave movies) can now communicate via scripting on the enclosing web page [Ip et. el.1997]. As we put the framework to real use, we recognize that there may be three different types of virtual apparatus, differentiated by the way instructional content is related to the functions provided:

- Some virtual apparatus are generic and content-free, for example a data logger. A data logger may continuously monitor the performance of a learner and store or pass on information for immediate feedback, assessment or evaluation purposes.
- Some virtual apparatus are content-rich, for example a simulation of a particular scenario. The virtual apparatus itself in this case contains the educational content. It may however, still be usefully operated with other apparatus, such as the data logger.
- Other virtual apparatus may be generic but content-dependent, such as a question shell object [See Fritze and Ip (this volume)]. These virtual apparatus are loaded with instructional content in order to operate.

While the first two types can be implemented as Version 1.0virtual apparatus, additional support is necessary for the third type [see our online resource for the details of the Version 1.0 specification, located at http://www2.meu.unimelb.edu.au/virtualapparatusframework/]. Version 1.1 is an extension to the virtual apparatus specification1.0 that defines a standard mechanism to support the generic but content-dependent virtual apparatus. Virtual apparatus

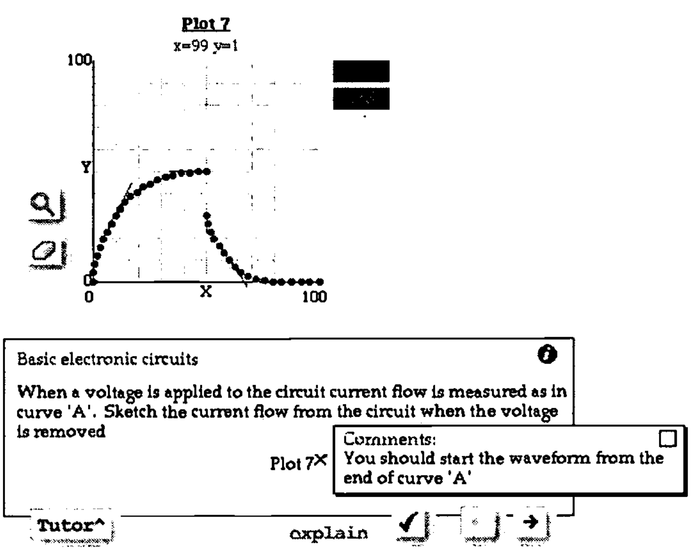
developed in accordance to version 1.0 and version 1.1 are inter-operable. The content is expressed in a tagged format and is referred here as VAScript.

## Typical use scenarios

### Learning Engine object model (with a master object)

The Learning Engines Project [Fritze and McTigue, 1997] outlines particular model of interactive object that can be independently produced and combined into useful learning activities. These objects correspond to visualisations, simulations, dialogue shells and interface extension components. The dialogue shell object takes the more important role, acting as a controlling agent that shapes the dialogue between the learner and other objects. The shell derives its content from a VAScript.

In the illustration below, we show two such components on a page. The bottom component is a particular dialogue shell, the Tutorial Item Set, which embodies a series of questions items specified by its current VAScript. The upper object is an interactive graphing object which can both plot curves and interpret those sketched by the student. The graph object has plotted an initial curve as specified by a <command> tag in the VAScript of the Item Set and relayed by the VAmessenger. The VAmessenger is the agent controlling inter-object communication within the VA Framework[Ip et. el. 1997]. A curve entered by the student has been reported back to the ItemSet which has judged it against the specified criteria and shown a corresponding feedback message. It is possible, if the author deems appropriate, to send a drawing command to the graph using the <command> tag in the ItemSet script to display the correct answer.

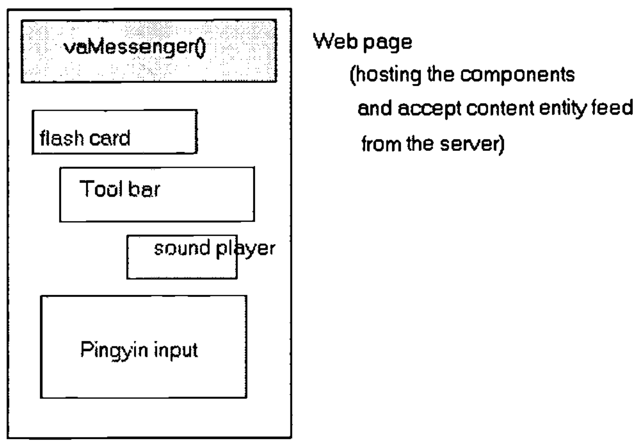



### NALSAS model (using peer-to-peer relationship)

Another project being undertaken for NALSAS (National Asian Languages and Studies in Australian Schools Taskforce) to teach Mandarin to teachers, assumes a different content model. All the components of a learning activity on the Web page communicate via the VAmessenger. No component is in a master position. The Flash Card, Sound Player and Pinyin Input software objects have been constructed as version 1.1 virtual apparatus and have their respective scripts. When the page is requested by a student, the server generates these

three scripts. When the student clicks on a "start" button on the tool bar, a message is sent by the tool bar apparatus to the VAmessenger. Logic written within the VAmessenger generates a command received by the three objects to use the first item in their respective scripts. The Flash Card displays a Chinese character briefly. The Sound Player plays the sound corresponding to the character and the Pinyin input is receives typed input from the student corresponding to the pinyin of the flashed character. After a correct input, the Pinyin Input sends a message to this effect and VAmessenger asks the other components to advance to the next element in their scripts. The synchronization is established by the index attribute in the second layer tags of the respective scripts.

An interesting point about the NALSAS project is that, while the flash card and sound player are shockwave movies, the tool bar and the pinyin input are Java applets. The VA framework provides the mechanism that enables them to inter-operate on the same page.




## Flexibility in adding data-logging functions as needed

A data-logger is an invisible apparatus. When activated by a line in the VAmessenger, a copy of all messages passing through the messenger can be relayed to it. In an online environment, the data-logger can establish a direct database connection to the server(e.g. via JDBC using a data logger written as a Java Applet). In an offline situation, another data-logger (e.g. written as an Active X control) writes the data to local storage. As long as both data-loggers have the same registered methods and properties, changing from online to offline delivery requires a change to just one line of code. In some situations, this can be automatically generated by the server side component.

## Adaptive content delivery using a back-end database server

The functionality provided by the virtual apparatus on the page can be easily customized by a content author or an instructional designer, for example in the flash card example above, by setting the umber of times the Chinese character is to be flashed. Content sent to the virtual apparatus as VAScript by the back-end database can be adapted according to the performance and/or background of the student. This flexibility is make practical by the principle of separating software function from content description.

# Technical Specification of VAScript

The Virtual Apparatus specification 1.1 concerns three parts:

- format of the educational content (i.e. VAScript).
- the compulsory parser behaviours to handle the VAScript. In different circumstances, a parser may exist within a virtual apparatus or externally as a generic parser apparatus.
- the support on the browser (using browser scripts) for VAScript.

VAScript is based on XML (Extended Markup Language). XML is a document specification offering significant advantages over HTML, particularly with its extensibility and DTD (Document Type Description) metadata descriptions and will soon replace HTML as the major Web document standard. The following is the technical specification of VAScript which relies on an understanding of XML.

## VAScript Syntax

A VAScript is a tagged document in the form of an XML document with the following notable features:

- every virtual apparatus supporting specification version 1.1 can have its own set of recognized tags. The definition of the tags, known as the DTD part of the VAScript, is also part of the specification.
- the DTD definition is now a part of the `VAinfo` and hence the function `getVAinfo` will return the DTD as well. `Vainfo` is a mechanism defined in Specification version 1.0 to enable content author to read the methods and properties of a virtual apparatus.
- the DTD defines the legal tags within the VAScript supported by the associated virtual apparatus. The meaning and/or the functions of the tags are specified as comments in accordance with the syntax requirement of XML document.
- In order to enable future compatibility with metadata, all the tags will begin with "VA:". Note the colon ":" is included. Effectively, all VAScript will operate within our own namespace.
- The DTD of the top element of a VAScript is:

  ```
  <!ELEMENT VAScriptName - - (SecondLevelTagName+) >
  <!ATTLIST VAScriptName
  LoadState (auto|preload|incremental) auto >
  ```

  (Note: If `LoadState=auto`, the VAScript will be loaded using the default behaviour of the virtual apparatus. If `LoadState=preload`, all the `SecondLevelTagName` will be loaded and made available at the same time. If `LoadState=incremental`, each `SecondLevelTagName` is treated as an element in a set. The virtual apparatus should provide mechanism to handle the `SecondLevelTagName` one by one.)

  See an explanation of the syntax of the DTD grammar at Appendix 1.
- The second level tag must be in the following form:

  ```
  <!ELEMENT SecondLevelTagName - - (OtherElements+)>
  <!ATTLIST SecondLevelTagName
  Index ID #IMPLIED>
  ```

  (Note: `Index` is necessary for all second level tags because on a web page, there may have several virtual apparatus interacting in order to provide a rich experience to the learner, these virtual apparatus rely on `index` to synchronize the content. As such, a server-side component should ensure that all `Index` on different VAScript on the

same page are mapped correctly. If Index is not supplied, it will be generated internally by the virtual apparatus, however it will not be available for use by the other virtual apparatus.)

- <EXTERNAL> </EXTERNAL>is always implicitly defined and is supported by the build-in parser. The <EXTERNAL> tag is defined as follows:
```
<!ELEMENT External - - (*)+>
<!ATTLIST External
target NAME #IMPLIED
TargetLoadState (discard|append) append
ExportIndex (UseSource|UseTarget|Increment|$ID)
UseSource>
```
  (Note: Any VAScript which are enclosed by the <EXTERNAL> </EXTERNAL> tags is called embedded VAScript. Embedded VAScript is not understood by the current parser and should be forwarded to the target virtual apparatus as specified in the target attribute. The embedded VAScript is a full VAScript with its own Top level element.
  Upon arrival at the target virtual apparatus, the embedded VAScript will replace the previously loaded VAScript at the target virtual apparatus if the TargetLoadState=discard. It will append to the previously loaded VAScript if the TargetLoadState=append.
  The Index in the second level element of the embedded VAScript will take the parent VAScript's current Index if ExportIndex=UseSource. The Index will be that specified in the embedded VAScript if ExportIndex=UseTarget. The index will be the numeric additive sum of the parent VAScript's current Index and the embedded VAScript Index if the ExportIndex=Increment. Finally, the index of the embedded VAScript will be ID if ExportIndex=$ID)
- <command> </command> is always implicitly defined and is supported by the build-in parser. The <command> tag is defined as follows:
```
<!ELEMENT command - - (*)+>
<!ATTLIST command
target NAME #IMPLIED>
```
  This is used to send command to the target virtual apparatus.

## Compulsory parameters supported by Version 1.1 virtual apparatus

Virtual apparatus conforming to virtual apparatus specification1.1 must also support the VAScriptURL parameter. This parameter provides the URL of the VAScript which maybe loaded during initialization. On successful loading, the virtual apparatus will send the message "VASLoadOK". Then the VAScript will be parsed. On successful parsing, the message "VASParseOK" is sent.

## Compulsory behaviours supported by Version 1.1 virtual apparatus

Virtual apparatus conforming to virtual apparatus specification1.1 must implement the following methods:

- SetVAScript(VAScript, LoadState, IndexState, InitIndex) where InitIndex is the first index in a set, replacing the index in the incoming

VAScript. `InitIndex` will be ignored if `IndexState=UseTarget`. If the `LoadState=Append`, the treatment of the index will depend on the `IndexState`.
This method will load the VAScript and parse accordingly. On successful parsing, send message "`ExtVASParseOK`"

- `SetVAScriptURL(VAScriptURL, TargetLoadState, ExportIndex, _ CurrentIndex)`
This method is similar to the previous method except it will initiate a loading of VAScript using the `VAScriptURL`. On successful loading, send message "`EXTVASLoadOK`". Then the VAScript will be parsed. On successful parsing, send message "`ExtVASParseOK`"

The built-in parser must support the following:

- When the `<external>` tag is encountered, the content of this tag is passed to `VASforward()` method in the browser environment.
- When the `<command>` tag is encountered, the content of this tag is passed to `VAScommandSwitch()` method in the browser environment.

**Browser Script support for Version 1.1**
When a page contains one or more virtual apparatus meeting this specification, it must implement:

- `VASforward(target, VAScript, TargetLoadState, ExportIndex, CurrentIndex)`
Where target is the name of the virtual apparatus to get the VAScript with the specified `TargetLoadState`, `ExportIndex` and `currentIndex`.
- `VAScommandSwitch(target, commandstring)`
Where target is the name of the virtual apparatus to receive the command specified in command string.

## Concluding remarks

The VAScript provides a way of applying XML, as a powerful document description language, to educational content in the form of modular software objects. While the Chinese flash card described may fall into the category of practice and drill, more sophisticated uses are also supported by the framework, such as with the Learning Engine components. The time and effort invested in creating the interactive graphing object, for example, can be better justified with its re-use in different subject areas by redesigning the scripts, rather than the object itself. At the same time, the object, as a virtual apparatus, can inter-operate with others through a communication standard offered by the VA framework.

## Reference:

[Ip and Canale, 96]Ip and Canale, 1996. "A model for authoring virtual experiments in web-based courses", presented at ASCILITE 96.
[Ip et.el. 1997] Ip, Canale, Fritze and Ji, 1997, "Enablingre-usability of courseware components with Web-based 'VirtualApparatus'" as presented at ASCILITE 97.
[Fritze and McTigue, 1997] Fritze P, McTigue P, 1997. "Learning Engines - A Framework For The Creation Of Interactive Learning Components On The Web" as presented at ASCILITE 97.

[Roschelle and Kaput, 1997] Roschelle J, Kaput J. 1997. "Educational Software Architecture and Systemic Impact: The Promise of Component Software" online http://www.simcalc.umassd.edu/simcalc/library/S&SImpact.hqx
[Rowley, 1995] Rowley, K. 1995."Understanding Software Interoperability in a Technology-supported System of Education", Cause/Effect Fall pp20-26

# NOTICE

## REPRODUCTION BASIS

- [x] This document is covered by a signed "Reproduction Release (Blanket) form (on file within the ERIC system), encompassing all or classes of documents from its source organization and, therefore, does not require a "Specific Document" Release form.

- [ ] This document is Federally-funded, or carries its own permission to reproduce, or is otherwise in the public domain and, therefore, may be reproduced by ERIC without a signed Reproduction Release form (either "Specific Document" or "Blanket").